

## **REMARKS**

In the Office Action, the Examiner rejected claims 1-4, 7-12, 15-20 and 23-28 under 35 U.S.C. § 102(b). The Examiner objected to claims 3, 6, 13, 14, 21, 22, and indicated that those claims would be allowable if rewritten in independent form.

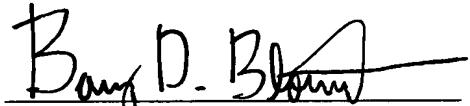
Based on the Examiner's indication of allowable subject matter, the Applicant has chosen to place the application in condition for allowance. Accordingly, the Applicant has amended independent claims 1, 9, 17 and 25 to include subject matter indicated as allowable by the Examiner. Accordingly, the Applicant has incorporated allowable subject matter of dependent claims 5, 13 and 21 into independent claims 1, 9, and 17, respectively. Further, the Applicant incorporated allowable subject matter of dependent claim 21 into independent claim 25. Additionally, dependent claims 5, 13, and 21 have been cancelled. It should be noted, however, that the Applicant does not concede the correctness of the rejections set forth in the Office Action. Moreover, the Applicant reserves the right to contest those rejections in a continuing application.

Upon entry of these amendments, the claims that remain pending in the present application are believed to be in condition for allowance. Therefore, the Applicant respectfully requests reconsideration and allowance of all pending claims.

**Conclusion**

In view of the remarks set forth above, the Applicant respectfully requests reconsideration of the Examiner's rejections and allowance of all pending claims 1-4, 6-12, 14-20 and 22-28. If the Examiner believes that a telephonic interview will help speed this application toward issuance, the Examiner is invited to contact the undersigned at the telephone number listed below.

Respectfully submitted,



Date: August 29, 2006

Barry D. Blount  
Reg. No. 35,069  
FLETCHER YODER  
P.O. Box 692289  
Houston, TX 77269-2289  
(281) 970-4545

**CORRESPONDENCE ADDRESS:**

Intellectual Property Administration  
Legal Department, M/S 35  
HEWLETT-PACKARD COMPANY  
P.O. Box 272400  
Fort Collins, CO 80527-2400

## **APPENDIX “A”**

[0064] FrameworkException is the other abstract subclass of WPAException and has the same types of constructors as FeatureException. The FrameworkException class may employ constructors similar to those discussed with respect to the FeatureException class. Each concrete subclass of FrameworkException should have an error code defined within it, and each time a new instance of that exception is created, the error code is passed to the appropriate superclass constructor.

[0065] BusinessException is one of the two abstract subclasses of FeatureException. Services must subclass this class for all business exceptions. Each concrete subclass should have an error code defined within it, and each time a new instance of that exception is created, the error code is passed to the appropriate superclass constructor. Because the class is abstract, it may have constructors that have the same format as FrameworkException and FeatureException.

[0066] SystemException is the other abstract subclass of FeatureException. Accordingly the SystemException constructors may follow the same format as BusinessException, FrameworkException, and FeatureException.

[0067] Each concrete subclass of WPAException should define an error code that is passed to the appropriate superclass constructor. Developers of web applications in accordance with embodiments of the present invention may use the following constructors when creating new exceptions, where ConcreteException should be substituted with the actual name of the concrete exception:

public ConcreteException() - This constructor is called when there is no additional information other than that included in the error catalog that should be written to the error log file

public ConcreteException(String errorMessage) - This is used when the programmer would like to log an additional message, perhaps to state exactly what occurred.

public ConcreteException(DiagnosticContext diagnosticContext) - Here, a diagnostic context object is added. The object should contain information about the context of the error, such as variable values, that can be used for debugging.

public ConcreteException(Exception caughtException) - This constructor is used when the ConcreteException is created as a result of another exception being caught. Using this constructor preserves the sequence of exceptions for possible output to the error trace log.

public ConcreteException(DiagnosticContext diagnosticContext, Exception caughtException) - When both a diagnostic context object and a caught exception are present, this constructor is called. It combines the added features of the previous two constructors.

**[0068]** WPAErrorActionForward is an abstract superclass for action forwards that are created when a page must be displayed to a user due to some caught exception. The following constructors may be used in conjunction with the WPAErrorActionForward class:

public WPAErrorActionForward(WPAException wpaException, HttpServletRequest request, String path) - The path parameter is the path to the page that should be displayed. It also obtains an instance to the TransactionProxy object.

public WPAErrorActionForward(String errorCode, HttpServletRequest request, String path) -

The path parameter is the path to the page that should be displayed. This constructor is used for interactions with other architectures, where an error result is returned. It also obtains an instance to the TransactionProxy object.

protected void logError() - This method calls the TransactionProxy's logError() method and stores the error data in the transaction for later logging.

protected void logBusinessActivityStatus() - This method stores the error code and success indicator for use in logging to the business activity log file.

protected abstract void log() - This method must be instantiated by a concrete subclass. It controls storing information for logging.

protected void setSuccessIndicator(int indicator) - This class is called by the subclasses to set the success indicator before the logBusinessActivityStatus() method is called.

public Object clone() - This method returns a copy of this WPAErrorActionForward instance.

**[0069]** WPABusinessErrorActionForward is a class that may be used to display an error page to the user after a business exception. The following constructors may be employed in conjunction with the WPABusinessErrorActionForward class:

public WPABusinessErrorActionForward(BusinessException e, HttpServletRequest request, String resultClassName, ActionMapping mapping) - This constructor enables the business

activity data to be extracted and logged. For this constructor, the WPAException must be a concrete instance of a BusinessException. The action mapping and the resultClassName are used to get the path of the page to be displayed.

`public WPABusinessErrorActionForward(String businessErrorCode, HttpServletRequest request, String resultClassName, ActionMapping mapping)` - This constructor is used for interactions with other architectures, where an error result is returned. This constructor enables the business activity data to be extracted and logged. For this constructor, the businessErrorCode is the error code of the business exception. The action mapping and the resultClassName are used to get the path of the page to be displayed.

`private void process()` - This method extracts the data from the exception and stores it in the ErrorInfoBean for possible access by JSP page. This method also calls the log method.

`protected void log()` - This method controls storing information for logging.

`public Object clone()` - This method returns a copy of this WPABusinessErrorActionForward instance.

**[0070]** An instance of a WPASystemErrorActionForward may be created when an error page should be displayed to a user because of an instance of a concrete subclass of SystemException or FrameworkException. The following constructors may be employed in conjunction with the WPASystemErrorActionForward class:

`public WPASystemErrorActionForward(SystemException wpae, HttpServletRequest request, ActionMapping mapping)` - This constructor enables the error data to be extracted and

logged. For this constructor, the WPAException must be a concrete instance of a SystemException. The ActionMapping object may be used in conjunction with a portal property specified in configuration resource file 152 (FIG. 2) to get the path of the error page to be displayed.

public WPASystemErrorActionForward(FrameworkException wpaе, HttpServletRequest request, ActionMapping mapping) - This constructor enables the error data to be extracted and logged. For this constructor, the WPAException must be a concrete instance of a FrameworkException. The ActionMapping object may be used in conjunction with a portal property specified in configuration resource file 152 (FIG.2 ) to get the path of the error page to be displayed.

public WPASystemErrorActionForward(String systemErrorCode, HttpServletRequest request, ActionMapping mapping) - This constructor is used for interactions with other architectures, where an error result is returned. This constructor enables the error data to be extracted and logged. For this constructor, the systemErrorCode is the error code of the system exception. The ActionMapping object may be used in conjunction with a portal property specified in configuration resource file 152 (FIG. 2) to get the path of the error page to be displayed.

private void processException() - This method extracts the data from the exception and stores it in the ErrorInfoBean for display to the user. This method also calls the log method.

private void processCode() - This method stores the error code in the ErrorInfoBean for display to the user. This method also calls the log method.

protected void log() - This method controls storing information for logging.

public Object clone() - This method returns a copy of this WPASystemErrorActionForward instance.

**[0071]** The WPASystemErrorActionForward may use the ErrorInfoBean object to store values that are later accessed by the error JSP page for display. There may be getter and setter methods for error code, UI message key, and title key.

**[0072]** The following discussion relates to the design of the error catalog 210 (FIG. 3). The error catalog 210 may provide exceptions with a centralized object that may be used to look up information by error code. Each service that creates its own concrete subclasses of SystemException must have an errorCode.properties file stored in an appropriate predetermined directory to define per SystemException error code the error name, error reason, and error recourse. Concrete subclasses of FrameworkException must also have similar error data recorded in an errorCode.properties file, but because those exceptions are created within the framework of a web presentation architecture constructed in accordance with embodiments of the present invention, they reside in the framework's errorCode.properties file. The numerical portion of the error codes should fall within a range that defines a general description of the error. The ranges may also indicate a title key and UI message key that should be used with each error when displaying an error page to the user. In addition, the type of error action forward may indicate a title key and UI message key that should be used in certain occasions when displaying an error page to a user. This information may be portal specific and portal information that may be stored in the a configuration file

such as the configuration file 212 (FIG. 3) may be used in conjunction with the error action forward name to indicate title keys and UI message keys.

**[0073]** As set forth above, the error catalog 210 may be populated when the controller 205 (FIG. 3) starts up (e.g. in the init() function of the controller 205). An ErrorCodeConfigurator may be among the configurators called during startup, and it may search for and open a configuration file for each service. Those of ordinary skill in the art will appreciate that a single configuration file may be used for storing information about all relevant services, if desired. The configuration file or files may be text properties files or the like. The properties are read and stored into an ErrorCodeMapper, then loaded into the ErrorCodeCatalog as ErrorCode objects or title and UI message data.

**[0074]** The functionality associated with the error catalog 210 (FIG. 3) may comprise a plurality of classes, which may be referred to as “actors.” Those actors may include the following:

WPACController - the controller (for example, the controller 205 (FIG. 3)).

WPACConfiguratorController - the controller class that calls all the configurators, given the list as defined in a predetermined configuration file.

ErrorCodeConfigurator - the configurator run during the controller initialization that opens the errorCode.properties file for each service.

ErrorCodeMapper - where the data from the errorCode.properties files of the services are stored.

Catalog - the abstract superclass of ErrorCodeCatalog.

ErrorCodeCatalog - this object stores ErrorCode objects, searchable by error codes. This object also stores title key and UI message key data.

ErrorCode - this object stores data about each exception: the error code, error name, error reason, and error recourse.

**[0075]** The following use case describes a typical course of events for a situation in which error data is loaded upon controller initialization:

7. During startup, the init() method of the controller makes the proper calls so the configurator classes are run (using the WPACConfiguratorController).
8. Among the configurator classes run is ErrorCodeConfigurator.
9. ErrorCodeConfigurator.configure() is called. This method uses the passed in ActionServlet parameter to obtain a servletContext. The servletContext is used to traverse the directory tree to find each service's errorCode.properties file.
10. Once found, the errorCode.properties file of each service is opened and the data is loaded into the singleton ErrorCodeMapper object.
11. ErrorCodeConfigurator then calls the configure() method of ErrorCodeCatalog.

12. For each property in the mapper, the catalog calls registerHandler, a method inherited from its superclass, Catalog. The registerHandler method creates an ErrorCode object for each entry that was an errorCode mapping in the properties file. These entries are then stored by error code in the “handlers” hashtable of the ErrorCodeCatalog. Additionally, the method isolates the uiMessage entries and stores them separately in a “uiMessages” hashtable the ErrorCodeCatalog.

**[0076]** The configuration file that is loaded by the controller at initialization may comprise three types of data: UI message key data, title key data and mapping data. The UI message key data may be defined as follows:

<range>.<portal\_name>=<UI\_Msg\_key>

or

<actionForwardName>.<portal\_name>=<UI\_Msg\_key>

where the range may take into account the numbering guidelines:

System unavailable	10000 to 10999
Service unavailable	15000 to 15999
Database related errors	20000 to 20999
File System errors	25000 to 25999
Service errors	30000 to 30999

and where UI\_Msg\_key comprises: <portal\_name>.error.<description\_of\_string> and where actionForwardName comprises the fully qualified pathname for one of WPASystemErrorActionForward or WPABusinessErrorActionForward.

**[0077]** An example set of UI message definitions follows:

```
10000-10999.example=example.error.system_unavail_msg
15000-15999.example=example.error.service_unavail_msg
20000-20999.example=example.error.db_err_msg
25000-25999.example=example.error.file_sys_err_msg
30000-30999.example=example.error.service_err_msg

com.samplePath.WPASystemErrorActionForward.example=example.error.service_unavail_
msg
com.samplePath.WPASystemErrorActionForward.example=example.error.service_unavail_
msg
```

**[0078]** Title keys that are stored in configuration files may be defined according to the following format:

<range>.title.<portal\_name>=<Title\_Msg\_key>

or

<actionForwardName>.title.<portal\_name>=<Title\_Msg\_key>

where the range takes into account the numbering guidelines described above and where Title\_Msg\_key may comprise: <portal\_name>.error.<description\_of\_string> (i.e. example.error.service\_err\_pg\_title), and where actionForwardName comprises the fully qualified pathname for one of WPASystemErrorActionForward or WPABusinessErrorActionForward.

**[0079]** An example follows:

```
10000-10999.title.example=example.error.system_unavail_pg_title
15000-15999.title.example=example.error.service_unavail_pg_title
20000-20999.title.example=example.error.db_err_pg_title
25000-25999.title.example=example.error.file_sys_err_pg_title
30000-30999.title.example=example.error.service_err_pg_title

com.samplePath.WPASystemErrorActionForward.title.example=example.error.service_unav
ail_pg_title
com.samplePath.WPASystemErrorActionForward.title.example=example.error.service_unav
ail_pg_title
```

[0080] The third type of data in the errorCode.properties file is the mapping of error code to error name, error reason, and error recourse. An exemplary format for that mapping follows:

errorCode=errorName1|errorReason1|Recourse

where the error code follows the format:

<portal>-pl-<service>-errorCode

[0081] An exemplary StartupException follows:

sampleFramework-pl-sampleFramework-10999=StartupClassInitializationError|Error  
occurred during initializing startup classes at weblogic startup|Check startup\_error.log file for  
any errors logged.

[0082] FIG. 5 is an object diagram of an architecture for object classes associated with an error catalog in accordance with embodiments of the present invention. The diagram is generally referred to by the reference numeral 400.

[0083] The ErrorCodeConfigurator class may implement the configure(ActionServlet) method defined in the Configurator interface. The ActionServlet parameter is used to locate and open the errorCode.properties files in the directory tree. The following method may be employed in conjunction with the ErrorCodeConfigurator class:

public void configure(ActionServlet servlet).

**[0084]** The ErrorCodeMapper class may store all the properties defined in the various service defined errorCode.properties files. It is a singleton object, and therefore can be called when the ErrorCodeCatalog is being configured. A singleton object is an object that exists in memory such that only one of that type of object exists at any time in memory. Once created, a singleton object is not destroyed after use, like most objects, but is kept in memory until accessed again. The following method may be employed in conjunction with the ErrorCodeMapper class:

public static ErrorCodeMapper getInstance() - Returns the instance of the ErrorCodeMapper.

**[0085]** The Catalog class is an abstract superclass of ErrorCodeCatalog. The registerHandler method that ErrorCodeCatalog inherits creates the ErrorCode objects from the data in the ErrorCodeMapper and stores them in a “handlers” hashtable. The title keys and UI message keys are also isolated and stored in a “uiMessages” hash table. If there is a problem encountered during this method, the FrameworkException RegisterHandleException is thrown. The following method may be employed in conjunction with the Catalog class:

public synchronized void registerHandler(String code, String value) throws RegisterHandleException.

**[0086]** The ErrorCodeCatalog class is a concrete subclass of Catalog that creates an error catalog in its configure() method. For each entry stored in the ErrorCodeMapper, registerHandler() is called, and either an ErrorCode object is created, or the entry is

recognized as a title key or UI message key entry. ErrorCode objects are stored in a hash table separate from the title and UI message hash table. The following methods may be employed in conjunction with the ErrorCodeCatalog class:

public synchronized static ErrorCodeCatalog getInstance() - ErrorCodeCatalog is a singleton object, so programmers may access it through its getInstance() method.

public void configure() - Creates the error catalog by accessing the ErrorCodeMapper and creating ErrorCode objects or storing title keys and UI message keys.

**[0087]** The ErrorCode object holds information about an exception, as loaded from the errorCode.properties file, including error code, error name, error reason, and error recourse. In addition to the constructor, there are getter functions for each of the fields just named. The following constructor may be employed in conjunction with the ErrorCode object:

public ErrorCode(String code, String name, String reason, String recourse).

**[0088]** While the invention may be susceptible to various modifications and alternative forms, specific embodiments have been shown by way of example in the drawings and will be described in detail herein. However, it should be understood that the invention is not intended to be limited to the particular forms disclosed. Rather, the invention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the invention as defined by the following appended claims.